Ein Ansatz zur Übertragung von Rangordnungen bei der Suche auf strukturierten Daten

Andreas Henrich und Günter Robbert
Universität Bayreuth, Fakultät für Mathematik und Physik,
Fachgruppe Informatik, 95440 Bayreuth
{andreas.henrich|guenter.robbert}@uni-bayreuth.de

Abstract: Ähnlichkeitsanfragen – oder allgemeiner Anfragen, die eine Rangordnung auf den Ergebnisdokumenten definieren – erlangen in Bereichen wie Multimedia oder Bioinformatik immer mehr an Bedeutung. Insbesondere im Bereich strukturierter Dokumente kommen dabei auch Anfragen vor, bei denen die Kriterien für die Rangordnung über verbundene Objekte definiert wird. So kann z.B. nach Bildern aufgrund einer Bedingung für den umgebenden Text gesucht werden. Steht nun eine Zugriffsstruktur für die Textblöcke bereit, so erscheint es vielversprechend, zunächst mit der Zugriffsstruktur ein Ranking der Textblöcke zu erstellen und dieses dann auf die Bilder zu überragen. Mit der Semantik dieser Übertragung und einem dazu anwendbaren Algorithmus beschäftigt sich die vorliegende Arbeit. Sie umfasst ferner eine Betrachtung verwandter Ansätze sowie die Präsentation experimenteller Ergebnisse.

1 Motivation

Bei klassischen Datenbankanwendungen haben Anfragen üblicherweise den Charakter harter Faktenbedingungen. Zum Beispiel wenn alle Aufträge des Kunden mit der Kundennummer 1234 gesucht werden. In anderen Anwendungsgebieten von Datenbanken, wie bei der Verwaltung multimedialer Dokumente oder in der Bioinformatik, sucht man dagegen oft nach den zu einem vage vorgegebenen Informationswunsch relevanten Dokumenten bzw. Objekten oder nach den zu einem vorgegebenen Musterobjekt ähnlichen Objekten. Hier wird ein Ordnungskriterium benötigt, das die Relevanz oder Ähnlichkeit der Objekte annähert und so ein Ranking der Objekte nach dem gewünschten Kriterium erlaubt. Im Normalfall werden dabei in der Antwort auf eine Anfrage nur die k relevantesten oder ähnlichsten Objekte erwartet.

Derartige „Ähnlichkeitsanfragen“ sind in der letzten Zeit in zahlreichen Arbeiten adressiert worden. Dabei sind einerseits Zugriffsstrukturen zur Sortierung der Daten nach einem einzelnen Ordnungskriterium und andererseits Algorithmen zur Kombination mehrerer jeweils nach einem Kriterium erstellter Rankings zu einem Gesamtranking vorgeschlagen worden – wir werden auf die wichtigsten Vorschläge hierzu in Abschnitt 3 genauer eingehen. Zugriffsstrukturen und Algorithmen, die eine gegebene Basismenge nach einem Ordnungskriterium sortieren, werden wir in dieser Arbeit als *Ranker* bezeichnen. *Ranker* arbeiten üblicherweise schrittweise nach einer *lazy evaluation* Strategie, die erlaubt, ihre

Ausgabe nach Art einer Pipe in UNIX nur so weit zu betrachten, wie dies in der konkreten Anwendung erforderlich ist. Typische Realisierungen von *Rankern* basieren auf mehrdimensionalen Zugriffsstrukturen – als Beispiele seien hier der M-tree [ZSAR98], der X-tree [BKK96] oder der LSD^h-tree [Hen98] genannt – oder invertierten Listen. Ein *Ranker* kann z.B. eingesetzt werden, um Bilder nach der Farbähnlichkeit im Hinblick auf ein Beispielbild sortiert auszugeben. Gerade bei Bildern gibt es aber neben der Farbähnlichkeit noch weitere relevante Ähnlichkeitskriterien wie die Texturähnlichkeit. Dieser Tatsache kann dadurch Rechnung getragen werden, dass man einen *Ranker* für die Farbähnlichkeit und einen *Ranker* für die Texturähnlichkeit einsetzt und die sich ergebenden Rangordnungen zu einem Gesamtranking verschmilzt. Für diese Aufgabe wurden Algorithmen wie Fagin's Algorithmus, Nosferatu oder Quick-Combine vorgeschlagen (vgl. Abschnitt 3). Wir werden Algorithmen, die mehrere Rangordnungen über Objekten der gleichen Grundgesamtheit zu einem kombinierten Ranking verschmelzen, als *Combiner* bezeichnen. Ein Aspekt, der bei diesen Ansätzen bisher aber nicht hinreichend betrachtet wurde, ist, dass gerade bei strukturierten multimedialen Dokumenten die zur Sortierung der gewünschten Objekte verwendeten Kriterien oft nicht für diese Objekte selbst sondern für mit diesen Objekten in einer bestimmten Beziehung (oder Relation) stehende Objekte definiert sind. Zwei Beispiele sollen dies verdeutlichen:

- Will man zu einem bestimmten Themengebiet relevante Bilder – ggf. auch Videos oder Audios – suchen, so kann man dieses Themengebiet häufig textuell besser beschreiben als z.B. durch ein Musterbild. Man kann nun ausnutzen, dass Bilder in strukturierten Dokumenten nicht isoliert sondern üblicherweise in einem textuellen Kontext vorkommen. Es gibt Textpassagen in der Nähe des Bildes und oft auch eine Bildunterschrift. Für diese Textobjekte im Umfeld der Bilder – was „Umfeld" im Einzelfall konkret bedeutet, ist natürlich zu spezifizieren – kann dabei mit Methoden des Information Retrieval ein Ranking nach ihrer Relevanz für das gegebene Themengebiet abgeleitet werden. Anschließend kann man versuchen, dieses Ranking auf die Bilder zu übertragen, um so die k „relevantesten" Bilder zu bestimmen.
- Ein anderes Beispiel ergibt sich, wenn wir Bilder suchen, die ein bestimmtes Logo enthalten. Hier ist es nicht sinnvoll, die vollständigen Bilder im Hinblick auf ihre Farb- oder Texturähnlichkeit mit dem gegebenen Logo zu vergleichen. Vielmehr müssen die Segmente der Bilder, die automatisch oder manuell gewonnen werden können, mit dem Logo verglichen werden. Wir erhalten so eine Ordnung auf den Bildsegmenten, die auf die Bilder selbst übertragen werden muss.

Das adressierte Problem kann damit allgemeiner formuliert werden: Gesucht werden Objekte des Typs ot_d (d für *desired*). Diese Objekte des Typs ot_d sind über eine wohldefinierte Beziehung rel_d (z.B. eine Fremdschlüsselbeziehung) mit Objekten des Typs ot_r (r für *related*) verbunden. Für die Objekte des Typs ot_r wird nun eine Relevanzordnung definiert, die auf die im Ergebnis gewünschten Objekte des Typs ot_d übertragen werden muss.

Zur Lösung dieses Problems schlagen wir vor, neben *Rankern* und *Combinern* sogenannte *Transferer* einzusetzen, die die „Übertragung" eines Rankings leisten. Es erscheint uns nützlich, diesen Transferprozess explizit zu machen und ihn nicht in den *Combinern* zu

integrieren, weil nur so die Semantik der Transferoperation und der entsprechende Algorithmus allgemeingültig angesprochen werden kann. *Ranker*, *Combiner* und *Transferer* arbeiten dabei als Produzenten und im Falle der *Combiner* und *Transferer* auch als Konsumenten von *„Objektströmen“*. Der Begriff des *Stroms* ist hier analog zur Verwendung des *Stream*-Begriffs in C++ oder Java im Sinne einer schrittweisen Erzeugung bzw. Anlieferung von Ergebnisobjekten zu verstehen. So können die initial von *Rankern* erzeugten Ströme beliebig als Eingabe für *Combiner* oder *Transferer* genutzt werden, deren Ausgaben wiederum in nachgeschaltete *Combiner* oder *Transferer* einfließen können. Dabei ist anzumerken, dass man sich neben *Rankern*, *Combinern* und *Transferern* natürlich noch weitere „Strom-verarbeitene“ Komponenten vorstellen kann. Zu denken ist z.B. an Filter, die aufgrund von Faktenbedingungen aus der Ausgabe eines *Rankers*, *Combiners* oder *Transferers* die Objekte entfernen, die die spezifizierte Bedingung nicht erfüllen.

Im vorliegenden Papier betrachten wir nun den Prozess der Übertragung eines Rankings aus verschiedenen Perspektiven. Zunächst ist die genaue Semantik der Übertragung zu klären. Wie kann z.B. die Relevanzbeurteilung für Textblöcke in der Umgebung eines Bildes sinnvoll auf das Bild übertragen werden? Wir werden dieser Frage in Abschnitt 4 nachgehen. Im Anschluss daran werden wir den grundlegenden Algorithmus für die Übertragung einer Relevanzordnung in Abschnitt 5 betrachten. In Abschnitt 6 werden wir der Frage nachgehen, wie sich der vorgeschlagene Ansatz in einer Benutzungsoberfläche zur Anfrageformulierung auf strukturierten Dokumenten niederschlagen kann. Schließlich werden wir in Abschnitt 7 Überlegungen zur Systemarchitektur angeben und erste experimentelle Ergebnisse vorstellen. Zuvor soll aber in Abschnitt 2 ein einführendes Beispiel gegeben und in Abschnitt 3 auf verwandte Ansätze Bezug genommen werden.

2 Ein einführendes Beispiel

Um unseren Ansatz so genau wie möglich beschreiben zu können, führen wir zunächst ein Beispielschema ein, das im Wesentlichen auf den Vorschlägen von Analyti et al. [AC97] basiert. Abbildung 1 zeigt dieses Schema.

Im oberen linken Bereich der Abbildung ist die Struktur der Dokumente modelliert. Wir nehmen an, dass ein Multimediadokument aus einer oder mehreren Unterstrukturen besteht, die wir im Folgenden als *„Chunks“* bezeichnen wollen. Diese *Chunks* bestehen ihrerseits aus atomaren Medienobjekten oder *Chunks* einer niedrigeren Ebene. Im unteren Bereich von Abbildung 1 sind die verschiedenen Subtypen zu Medienobjekten dargestellt – nämlich *Image*, *Video*, *Audio* und *Text*. Der dritte Teil unseres Beispielschemas, der im oberen rechten Bereich zu sehen ist, repräsentiert die potentielle Segmentierung der Medienobjekte. So könnte ein Bild in die Bereiche segmentiert sein, die die konzeptuellen Objekte repräsentieren – hier sprechen wir von einer räumlichen Segmentierung – oder ein Video könnte in einzelne Szenen aufgeteilt sein – in diesem Fall sprechen wir von einer zeitlichen Segmentierung.

Ausgehend von diesem Schema können wir eine Anfrage formulieren, die die in der Einleitung angeführten Szenarien aufnimmt und als typisches Anwendungsbeispiel für den in

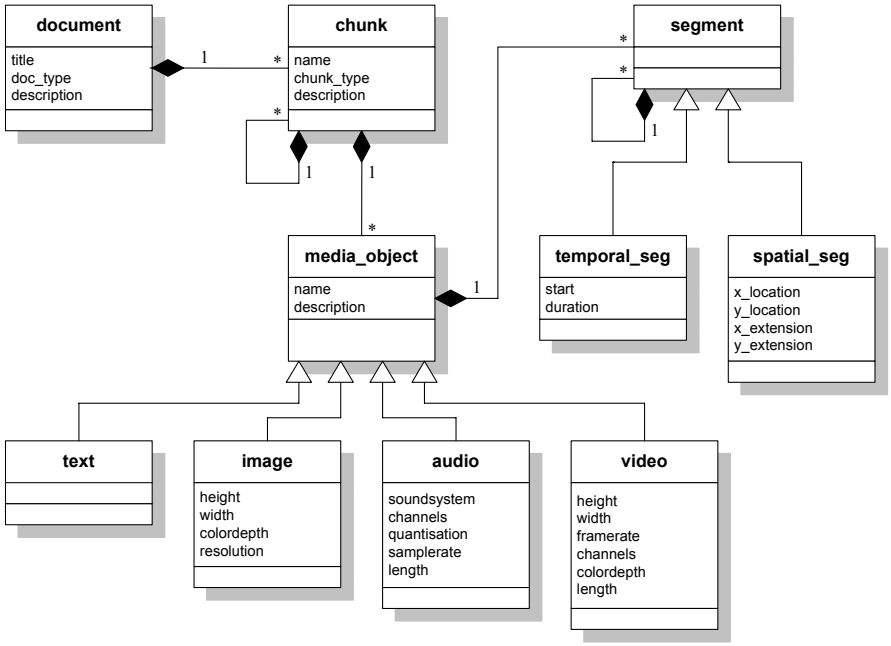

Abbildung 1: Ein Beispielschema für multimediale Dokumente

Abschnitt 5 vorgestellten *RSV-Transfer*-Algorithmus angesehen werden kann. Wir gehen dazu von einem Anwender aus, der nach Bildern sucht, die ein bestimmtes Logo enthalten und in deren Nähe der Text von *Skifahren* oder allgemeiner von *Wintersport* handelt. In diesem Fall sind die gewünschten Objekte Bilder. Zwei Ordnungskriterien sind definiert: (1) *Das Bild soll ein gegebenes Logo enthalten.* Diese Bedingung kann als Ähnlichkeitsbedingung für die dem Bild zugeordneten Segmente formuliert werden. Wir suchen daher nach einem Bild, für das eines der zugeordneten Bildsegmente im Hinblick auf Farbe, Textur und ggf. Form ähnlich zu dem vorgegebenen Logo ist. (2) *Der Text in der Umgebung des Bildes soll sich mit „Skifahren“ oder allgemeiner „Wintersport“ beschäftigen.* Ausgehend von unserem Beispielschema können wir den Text in der Nähe eines Bildes als die Menge der Textobjekte definieren, die im gleichen *Chunk* wie das Bild enthalten sind. Wir können dabei z.B. das Vektorraummodell [Sal89] verwenden, um diese Textobjekte im Hinblick auf ihre Ähnlichkeit zum Anfragetext „Skifahren oder Wintersport“ zu ordnen.

Durch diese beiden Ordnungskriterien haben wir nun ein Ranking der Bildsegmente und ein Ranking der Textobjekte und wir müssen aus diesen beiden Rangordnungen ein Ranking für die gesuchten Bildobjekte selbst ableiten. Abbildung 2 verdeutlicht, wie dies mit Hilfe von *Rankern*, *Combinern* und *Tranferern* geschehen kann.

Zunächst werden in Schritt (1.) mit Hilfe entsprechender *Ranker* vier initiale Ordnungen erstellt. Wir gehen dabei davon aus, dass für die Bildsegmente im Hinblick auf die Ähnlichkeit zum gegebenen Logo drei unterschiedliche Ähnlichkeitskriterien angewendet werden. Die drei sich hieraus ergebenden Rankings werden in Schritt (2.) zu einem ge-

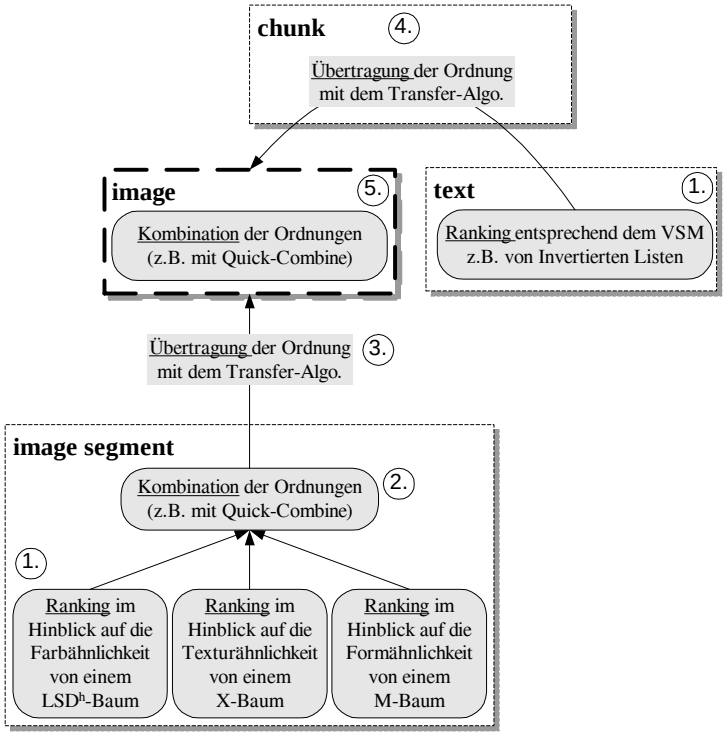

Abbildung 2: Ablauf der Bearbeitung der Beispielanfrage

meinsamen Ranking für die Bildsegmente kombiniert. Damit liegen die Rangordnungen für die Textdokumente und die Bildsegmente vor. Diese Ordnungen werden in den Schritten (3.) und (4.) mit dem *RSV-Transfer*-Algorithmus auf die Bilder übertragen. Schließlich werden in Schritt (5.) die beiden sich hieraus ergebenden Rangordnungen für die Bilder kombiniert.

An dieser Stelle sei angemerkt, dass man sich im obigen Beispiel natürlich auch eine speziell auf diese Anfrage zugeschnittene Zugriffsstruktur vorstellen kann. Eine solche „Speziallösung“ scheitert aber zwangsweise schon an geringfügig modifizierten Anfragen. Unser Ansatz ist dagegen durch die Kombination von *Rankern*, *Combinern* und *Transferern* flexibel auf ein breites Spektrum von Anfragen anwendbar.

3 Ein Überblick über Verfahren der Ähnlichkeitssuche

Wie bereits in der Motivation erwähnt, wurden in den letzten Jahren zum Thema Ähnlichkeitssuche eine Vielzahl von Artikeln publiziert. Die hinsichtlich unseres Ansatzes relevante Literatur lässt sich in die folgenden Kategorien unterteilen:

Verfahren zur Berechnung von Ähnlichkeitsanfragen

Im Rahmen dieses Forschungsgebietes wurden und werden Verfahren entwickelt, die als Ergebnis einer Ähnlichkeitsanfrage eine anhand eines vorgegebenen Ordnungskriteriums sortierte Liste (eine Rangordnung) von Elementen zurückliefern. Hierbei interessieren den Informationssuchenden meist nur die ersten k relevanten Treffer des Ergebnisses. Deshalb beschäftigt sich ein Forschungszweig explizit mit dem Thema der möglichst effizienten Berechnung der ersten k-Treffer von Ähnlichkeitsanfragen (Top-k Anfragen). Hierbei werden zwei unterschiedliche Aspekte betrachtet:

Im ersten Bereich wird das Problem aus der Perspektive der Anfrageoptimierung unter anderem in objektrelationalen Datenbanken betrachtet. In den Arbeiten von Carey und Kossmanns [CK98, CK97] wird aufgezeigt, wie ein STOP-AFTER-Operator, der zur Angabe der Kardinalität des Ergebnisses bei der Anfrageformulierung dient, in ein Datenbanksystem integriert werden kann. Insbesondere wird dargelegt, welche Strategien bei der internen Anfragebearbeitung zum Einsatz kommen können, um Anfragen unter Verwendung des STOP-AFTER-Operators effizient abarbeiten zu können. Eine andere Möglichkeit zur Optimierung wird in den Artikeln von Chaudhuri und Gravano [BCG02] beschrieben. Hier wird basierend auf Datenbankstatistiken versucht eine Top-k Anfrage mittels geeigneter Anfrageumformulierungen so auf komplexe Bereichsanfragen auf ein Datenbanksystem abzubilden, dass nur ein möglichst kleiner Teil des Datenbestandes zur Berechnung des Anfrageergebnisses herangezogen werden muss.

Im zweiten Bereich geht es um die Entwicklung spezieller Zugriffs- oder Indexstrukturen, welche Ähnlichkeitsanfragen effizient unterstützen. Für diesen Zweck wurden zahlreiche Ansätze und Implementierungen vorgestellt. Häufig wurden hierzu mehrdimensionale Bäume eingesetzt, wie beispielsweise der M-tree [ZSAR98], der X-tree [BKK96] oder der LSD^h-tree [Hen98]. Andere Realisierungen verwenden als Indexstruktur invertierte Listen [SMMR99] oder basieren auf einem schnellen sequentiellen Durchlauf der verwalteten Daten, wie z.B. bei den VA-Files [WSB98]. Im Sinne unseres Sprachgebrauchs handelt es sich hier um mögliche Implementierungen von *Rankern*.

Verfahren zur Kombination von Rangordnungen

In der Motivation haben wir aufgezeigt, dass es Anwendungsfälle gibt, in denen die Kombination mehrerer Rangordnungen zu einer Gesamtordnung notwendig ist. Hierzu wurden effiziente Verfahren zur Kombination von Rangordnungen entwickelt. Diese Verfahren gewinnen in letzter Zeit stark an Bedeutung, da sie nicht nur bei der Verschmelzung mehrerer Rangordnungen, die von einem einzelnen Datenbanksystem geliefert werden, anwendbar sind. Vielmehr finden solche Algorithmen zunehmend auch bei der Berechnung von Gesamtergebnissen über verteilte Datenkollektionen hinweg Anwendung. Problematisch sind dabei aber – insbesondere wenn Rangordnungen von heterogenen Systemen kombiniert werden sollen – die unterschiedlichen Zugriffsmöglichkeiten auf die einzelnen Rangordnungen, die die Datenbanksysteme anbieten, sowie die Ermittlung des Gesamtrankings. Diese Problematik wurde schon vor einigen Jahren erkannt und als *Collection-Fusion-Problem* bezeichnet [VGJL94].

Zwischenzeitlich wurde eine Vielzahl von Kombinationsalgorithmen publiziert, die vom Arbeitsprinzip her ähnlich vorgehen, sich aber durch unterschiedliche Anforderungen und

Anforderung/ Verfahren	wahlfreier Zugriff	sortierter Zugriff	inkrementell anwendbar	gleiche Objekt-IDs
Buckley-Lewitt	nicht notwendig	nicht notwendig	nein	ja
FA	erforderlich	erforderlich	nein	ja
TA	erforderlich	erforderlich	nein	ja
NRA	nicht notwendig	erforderlich	nein	ja
NRA-RJ	nicht notwendig	erforderlich	ja	ja
J^*	nicht notwendig	erforderlich	ja	nein
Nosferatu	nicht notwendig	erforderlich	ja	ja
Quick-Combine	erforderlich	erforderlich	nein	ja
Stream-Combine	nicht notwendig	erforderlich	ja	ja
Rank-Combine	nicht notwendig	erforderlich	ja	ja

Tabelle 1: Anforderungen und Eigenschaften der jeweiligen Kombinationsverfahren

Eigenschaften auszeichnen. In Tabelle 1 werden die wesentlichen Eigenschaften und Anforderungen der im Weiteren aufgeführten Verfahren gegenübergestellt. Wir untersuchen hierbei, welche Arten des Zugriffs von den eingehenden Rangordnungen unterstützt werden müssen, damit die Kombinationsalgortihmen anwendbar sind. Wir unterscheiden zwischen zwei Arten des Zugriffs, dem sortierten und dem wahlfreien Zugriff. Als sortierter Zugriff wird ein Zugriff bezeichnet, bei dem nacheinander in sortierter Reihenfolge Elemente aus einer Rangordnung entnommen werden. Von einem wahlfreien Zugriff sprechen wir, wenn zu einer vorgegebenen Objekt-ID der so genannte *Retrieval-Status-Wert* (vgl. Abschnitt 4) des zugehörigen Objekts in einer Rangordnung gesucht wird. Des weiteren wird in Tabelle 1 aufgezeigt, ob ein Verfahren inkrementell eingesetzt werden kann, d.h. ob weitere Verarbeitungsschritte auf höherer Ebene erfolgen können bevor die Berechnung der Gesamtliste abgeschlossen ist. Dies hat zur Folge, dass bei inkrementellen Verfahren der Wert k nicht vor Beginn der Berechnung bekannt sein muss.

Als weitere Eigenschaft untersuchen wir, ob ein Algorithmus die Kombination von Rangordnungen auch dann ermöglicht, wenn die in den zu kombinierenden Ordnungen enthalten Objekt-IDs disjunkt sind. Hierbei zeigt sich, dass viele Verfahren nur dann einsetzbar sind, wenn die Kombination über nicht disjunkte Mengen von Objekt-IDs erfolgen kann. Dies hat zur Folge, dass solche Verfahren Gesamtordnungen nur bestimmen können, wenn in den Eingangsordnungen die gleichen Objekt-IDs enthalten sind, wobei die Objekt-IDs natürlich an unterschiedlichen Stellen der verschiedenen Ordnungen auftreten können.

Einer der ersten Algorithmen zur Kombination von Rangordnungen wurde von Buckley und Lewit [BL85] im Zusammenhang mit invertierten Listen publiziert. Dieser Algorithmus benötigt weder einen wahlfreien noch einen sortierten Zugriff auf die Eingangslisten sondern lediglich einen Zugriff auf die Listenelemente in beliebiger Folge. Er ist nur sehr beschränkt inkrementell anwendbar. Grundlegende Arbeiten zum Thema Kombination von Rangordnungen wurden von Fagin geleistet, der mit dem Algorithmus FA (Fagin's Algorithm) [Fag98] ein effizientes Kombinationsverfahren vorstellte. Dieser Algorithmus benötigt für die Verarbeitung sowohl den sortierten als auch den wahlfreien Zugriff. Später folgte eine verbesserte Version von FA, TA (Threshold Algorithm) [FLN01]

genannt. TA verwendet ebenfalls den sortierten und den wahlfreien Zugriff für die Berechnung der Gesamtordnung. Sowohl FA als auch TA können nicht inkrementell angewendet werden. Die Algorithmen Quick-Combine [GBK00] und Multi-Step [NR99] basieren auf Fagin´s Algorithmus TA und unterscheiden sich im Wesentlichen durch verbesserte Abbruchbedingungen. Im Gegensatz zu den genannten Algorithmen benötigen die Verfahren NRA [Fag98] und Nosferatu [PP97] keinen wahlfreien Zugriff, NRA ist in der publizierten Originalversion im Vergleich zu Nosferatu jedoch nicht inkrementell einsetzbar. Der Algorihtmus NRA-RJ [IAE02] stellt eine Verbesserung des NRA-Algorithmus dar, der inkrementell eingesetzt werden kann. Weitere interessante Algorithmen für die Kombination von Rangordnungen sind RankCombine [HR01b], StreamCombine [GBK01], und J* [NCS$^+$01]. Alle drei Verfahren benötigen keinen wahlfreien Zugriff und sind zudem inkrementell einsetzbar. Nur der J* Algorithmus unterstützt die Kombination von Rangordnungen im Falle disjunkter Objekt-IDs. Mehr noch erlaubt J* die Übertragung von Ordnungen eines Objekttyps auf einen anderen Objekttyp, da hier mit Hilfe von benutzerdefinierten Prädikaten gesteuert werden kann, wie die verschiedenen Listen zu kombinieren sind. Die Übertragung der Rangordnungen erfolgt beim J* Algorithmus jedoch implizit bei der Kombination und weder die Semantik noch der Zusammenhang zwischen Kombination und Übertragung werden von den Autoren angesprochen. Im Gegensatz dazu verfolgen wir in dieser Arbeit den Ansatz, die Übertragung einer Rangordnung explizit zu betrachten und von ggf. vor- oder nachgelagerten Kombinationen von Rangordnungen zu trennen.

4 Die Semantik der Übertragung einer Ordnung

Bevor wir in Abschnitt 5 den Algorithmus zur Übertragung einer Rangordnung von einem Objekttyp auf einen verbundenen Objekttyp vorstellen, müssen wir zunächst die denkbaren Semantiken für diese Übertragung betrachten. Hierzu greifen wir auf die in Abschnitt 2 eingeführte Beispielanfrage zurück und vereinfachen sie zunächst indem wir nur die Suche nach Bildern betrachten, die ein gegebenes Logo enthalten. Die Situation ist hier wie folgt: Zunächst wird für die Bildsegmente eine Rangordnung berechnet. Dazu wird für die einzelnen Bildsegmente ein so genannter *Retrieval-Status-Wert* berechnet, der sich z.B. aus dem Vergleich der Farbhistogramme des Bildsegmentes und des Anfragelogos oder durch einen *Combiner* als gewichtete Kombination der Farb-, der Textur- und ggf. der Formähnlichkeit ergeben kann. Diese Retrieval-Status-Werte definieren nun zwar eine Rangordnung auf den Bildsegmenten, wir sind aber an einem Ranking der Bilder selbst interessiert. Daher ergibt sich die Notwendigkeit, für jedes Bild einen abgeleiteten Retrieval-Status-Wert auf Basis der Retrieval-Status-Werte der ihm zugeordneten Bildsegmente zu berechnen, um die gewünschte Ordnung auf den Bildern zu definieren.

$RSV_r(ro)$ sei der Retrieval-Status-Wert (*retrieval status value*) des Objekts ro (ro für „related object“ und RSV_r für den RSV eines verbundenen (*related*) Objekts). In unserem Beispiel wäre ro ein Bildsegment. Des weiteren sei $\{ro_{i,1}, ro_{i,2}, \ldots, ro_{i,n_i}\}$ die Menge der verbundenen Objekte die mit dem gewünschten (*desired*) Objekt do_i in Beziehung stehen. In unserem Beispiel würde diese Menge alle Bildsegmente enthalten, die dem Bild

do_i zugeordnet sind. Schließlich wollen wir annehmen, dass hohe Retrieval-Status-Werte besonders relevante Objekte identifizieren. Dann benötigen wir eine Funktion $\mathcal{F}$, die den abgeleiteten Retrieval-Status-Wert $RSV_d(do_i)$ von den mit do_i verbundenen Objekten und ihren Retrieval-Status-Werten ableitet:

$$RSV_d(do_i) \stackrel{\text{def}}{=} \mathcal{F}\,(\,\langle ro_{i,1}, RSV_r(ro_{i,1})\rangle,\ \ldots,\ \langle ro_{i,n_i}, RSV_r(ro_{i,n_i})\rangle\,)$$

Wir wollen nun einige für die Praxis relevante Beispiele für die Funktion $\mathcal{F}$ betrachten:

- *Der maximale RSV_r Wert*

  In diesem Fall vereinfacht sich die Funktion zu $RSV_d(do_i) \stackrel{\text{def}}{=} \max\{RSV_r(ro_{i,1}), \ldots, RSV_r(ro_{i,n_i})\}$. Dies bedeutet, dass der Retrieval-Status-Wert des gewünschten Objekts do_i durch das verbundene Objekt mit dem höchsten Retrieval-Status-Wert definiert wird. In unserem Beispiel würde dies dazu führen, dass das dem Logo ähnlichste Bildsegment den Retrieval-Status-Wert des gesamten Bildes definiert.

  Nun muss noch die Berechnung für Objekte geklärt werden, zu denen keine verbundenen Objekte existieren. In diesem Fall sollte $RSV_d(do_i)$ zum kleinsten denkbaren Retrieval-Status-Wert definiert werden – typischerweise null. Dies ist auch für die folgenden alternativen Definitionen der Funktion $\mathcal{F}$ eine sinnvolle Wahl.

- *Der durchschnittliche RSV_r Wert*

  Der Durchschnitt über alle RSV_r Werte ist ein anderes Beispiel für eine mögliche Semantik: $RSV_d(do_i) \stackrel{\text{def}}{=} \frac{1}{n_i} \cdot \sum_{j=1}^{n_i} RSV_r(ro_{i,j})$. Eine nähere Betrachtung ergibt allerdings, dass die Bedeutung des Durchschnitts stark vom angewendeten Ähnlichkeitsmodell abhängt. Im Hinblick auf unser Beispiel könnte der Retrieval-Status-Wert eines Bildsegmentes verglichen mit dem gegebenen Logo zum Beispiel über eine normierte Farbähnlichkeit definiert sein. Die Normierung würde dazu führen, dass Segmente unterschiedlicher Größe trotzdem den gleichen Einfluss auf die Ähnlichkeitssortierung der Bilder haben. Analoge Situationen können auftreten, wenn verbundene Textobjekte adressiert werden und das Vektorraummodell mit einer Dokumentlängennormierung angewendet wird. In diesen Fällen erscheint daher die Anwendung eines „gewichteten Durchschnitts“ ggf. angemessener.

- *Ein gewichteter Durchschnitt*

  Um einen gewichteten Durchschnitt zu berechnen, benötigen wir eine Definition der Größe (size) eines Objekts ro_i. Für Bildsegmente könnte diese Größe über die Zahl der Pixel und für Textobjekte über die Zahl der Worte definiert werden. Ausgehend von einer solchen Größendefinition können wir folgende Semantik definieren:

  $$RSV_d(do_i) \stackrel{\text{def}}{=} \sum_{j=1}^{n_i} RSV_r(ro_{i,j}) \cdot \frac{\mathsf{size}(ro_{i,j})}{\sum_{h=1}^{n_i} \mathsf{size}(ro_{i,h})}$$

  Für einen breiten Bereich von Ähnlichkeitskriterien bedeutet diese Definition, dass letztlich die Vereinigung über alle verbundenen Objekte $\left(\bigcup_{j=1}^{n_i} ro_{i,j}\right)$ betrachtet

wird, um den Retrieval-Status-Wert für do_i zu berechnen. Für Bilder mit verbundenen Bildsegmenten bedeutet dies, dass das Bild als Vereinigung der verbundenen Segmente betrachtet wird. Für einen Abschnitt mit Textobjekten bedeutet es, dass der Abschnitt als Konkatenation der verbundenen Textobjekte betrachtet wird.

- *Der minimale RSV_r Wert*

  In einigen Situationen kann es auch sinnvoll sein, die Ähnlichkeit des gewünschten Objekts über das „unähnlichste" verbundene Objekt zu definieren. In diesem Fall ergibt sich $RSV_d(do_i) \stackrel{\text{def}}{=} \min\{RSV_r(ro_{i,1}), \ldots, RSV_r(ro_{i,n_i})\}$. Hier wird der Retrieval-Status-Wert für do_i über die „am schlechtesten passende" Komponente bestimmt. Ein Beispiel, in dem dies sinnvoll sein könnte, ergibt sich, wenn im Hinblick auf die verbundenen Objekte sehr homogene Objekte do_i gesucht werden.

Neben den oben angeführten Semantiken existieren weitere Möglichkeiten. So könnte der Median oder ein α-Perzentil in entsprechenden Situationen sinnvoll sein. Im Folgenden werden wir sehen, dass unser Ansatz für die Übertragung einer Ordnung auch bei solchen Definitionen anwendbar bleibt, sofern die Semantik sicherstellt, dass $RSV_d(do_i) \leq \max\{RSV_r(ro_{i,1}), \ldots, RSV_r(ro_{i,n_i})\}$ gilt. Für $\mathcal{F}$ scheiden damit lediglich Funktionen wie die Summe oder das Produkt aus. Ob der Einsatz solcher Aggregationsfunktionen im Rahmen von Ähnlichkeitsanfragen sinnvolle Anwendungen hat, ist derzeit nicht klar und Gegenstand unserer aktuellen Forschungsarbeiten.

5 Der RSV-Transfer Algorithmus

Nachdem wir die Semantik der Übertragung eines Ähnlichkeitskriteriums betrachtet haben, können wir uns dem entsprechenden Algorithmus zuwenden. Das Problem, welches bei der Übertragung einer Rangordnung gelöst werden muss, kann wie folgt beschrieben werden: Wir betrachten eine Anfrage, die eine Ordnung für Objekte eines gewünschten Typs ot_d fordert (z.B. für Bilder). Allerdings ist die Ordnung nicht für die Objekte des Typs ot_d definiert, sondern für verbundene Objekte des Typs ot_r (z.B. Bildsegmente).

Wir nehmen nun an, dass die *„Verbindung"* (oder *„Relation"*) zwischen diesen Objekten wohldefiniert ist (z.B. durch einen Pfadausdruck) und dass sie bidirektional traversiert werden kann. Dies bedeutet, dass wir das betroffene Objekt – oder die betroffenen Objekte – des Typs ot_d für ein Objekt vom Typ ot_r und die verbundenen Objekte vom Typ ot_r für ein Objekt vom Typ ot_d bestimmen können. In unserem Beispiel bedeutet dies, dass wir für jedes Bildsegment sein Ursprungsbild und für jedes Bild die zugeordneten Bildsegmente (effizient) ermitteln können. In unserem konkreten Beispiel wird es nur ein Ursprungsbild für jedes Bildsegment geben. Es sind aber Situationen denkbar, in denen ein verbundenes Objekt mehreren Objekten des Typs ot_d zugeordnet ist. Die konkreten Charakteristika der Traversierungsoperationen in beiden Richtungen hängen offensichtlich von der Datenbank oder dem Objektspeicher ab, der zur Verwaltung der Dokumente eingesetzt wird. Bei objektrelationalen Datenbanken erlauben z.B. Join-Indizes und Indexstrukturen für Nested Tables ein hinreichend effizientes Traversieren der Verbindungen.

Zusätzlich nehmen wir an, dass eine Eingabeordnung (z.B. aus einem *Ranker* oder *Combiner*) gegeben ist, die eine Sortierung der Objekte des Typs ot_r liefert.

Ausgehend von diesen Annahmen, kann der Übertragungsalgorithmus wie folgt arbeiten: Er benutzt die Rangordnung – bzw. in Anlehnung an die Begrifflichkeiten von C++ oder Java den *Strom* – mit den geordneten Objekten vom Typ ot_r als Eingabe. Für die Elemente, die schrittweise aus diesem Eingabestrom entnommen werden, wird das betroffene Objekt – oder die betroffenen Objekte – vom Typ ot_d durch eine Traversierung der entsprechenden Beziehungen ermittelt. Nun werden die RSV_d Werte für diese Objekte vom Typ ot_d entsprechend der gewählten Semantik bestimmt und das aktuell betrachtete Objekt vom Typ ot_d wird in eine Hilfsdatenstruktur eingefügt, in der alle bisher betrachteten Objekte gemeinsam mit ihren RSV_d Werten verwaltet werden. Nun wird das nächste Objekt vom Typ ot_r aus dem Eingabestrom entnommen und untersucht; ist der RSV_r Wert dieses Objekts kleiner als der höchste RSV_d Wert eines Elements in der Hilfsdatenstruktur, das noch nicht ausgegeben wurde, so wird dieses Element aus der Hilfsdatenstruktur nun im Ausgabestrom des Übertragungsalgorithmus ausgegeben.

Für eine detailliertere Betrachtung des Algorithmus müssen wir die Charakteristika der Hilfsdatenstruktur festlegen, die wir mit AL (*auxiliary list*) bezeichnen wollen. AL verwaltet die vorläufigen Ergebnisobjekte zusammen mit ihren RSV_d Werten. Genauer verwaltet AL Paare der Form $\langle do_i; RSV_d(do_i)\rangle$ mit type$(do_i) = ot_d$. Diese Paare sind nach den RSV_d Werten absteigend sortiert. Für die Hilfsdatenstruktur AL werden nun die folgenden Operationen benötigt: createAL() erzeugt eine leere Hilfsdatenstruktur. getObj(AL, i) liefert das Objekt, dessen RSV_d Wert unter den Objekten in AL Rang i einnimmt. getRSV(AL, i) liefert den RSV_d Wert für das Objekt, dessen RSV_d Wert unter den Objekten in AL Rang i einnimmt. contains(AL, do_j) prüft, ob es in AL einen Eintrag für do_j gibt. insert$(AL, \langle do_l; RSV_d(do_l)\rangle)$ fügt einen Eintrag für do_l in AL ein. Dabei wird die Sortierung im Hinblick auf die RSV_d Werte erhalten. Existieren mehrere Objekte mit dem gleichen RSV_d Wert, so wird der neue Eintrag hinter die anderen Einträge mit gleichem Wert eingereiht. size(AL) liefert die Anzahl der Einträge in AL.

Ausgehend von diesen Definitionen können wir eine Klasse *Transferer* angeben, die einen Konstruktor und eine getNext-Methode zur Verfügung stellt. Diese Klasse ist in einer programmiersprachlichen Notation in Abbildung 3 angegeben.

Die Attribute, die für ein *Transferer*-Objekt verwaltet werden müssen, umfassen den Eingabestrom, eine Definition der gewünschten Beziehung zwischen den Objekten des Typs ot_d und ot_r, die Hilfsdatenstruktur AL, eine Variable o_r, die das Element des Eingabestroms aufnimmt, welches als nächstes zu bearbeiten ist und die Anzahl der bisher in den Ausgabestrom gestellten Objekte. Der Konstruktor initialisiert diese Variablen und liest das erste Objekt aus dem Eingabestrom. Dieses Objekt wird in der Variable o_r abgelegt. Die getNext-Methode arbeitet wie folgt:

- So lange der Eingabestrom nicht vollständig abgearbeitet ist ($o_r \neq \perp$) und entweder alle aktuell in AL verwalteten Objekte bereits ausgegeben wurden (size$(AL) < k$) oder der RSV_r Wert von o_r größer als der größte RSV_d Wert eines noch nicht ausgegebenen Objekts in *AL* ist, wird das aus dem Eingabestrom stammende Objekt o_r betrachtet und das nächste Element aus dem Eingabestrom nach o_r gelesen.

```
Class Transferer {
  Stream : inputStream;
  RelationshipDef : rel_d; /* Beziehung zu den verbundenen Objekten */
  AuxiliaryList : AL;
  InputObject : o_r; /* das nächste Objekt, das betrachtet werden müßte */
  Integer : k; /* Nummer des nächsten in der Ausgabe zu liefernden Objekts */

  constructor(Stream : input, RelationshipDef : rel) {
    inputStream := input;  rel_d := rel;
    AL := createAL();
    o_r := streamGetNext(inputStream);
    if o_r = ⊥ then exception(„leerer Eingabestrom“);
    k := 1;
  }

  getNext() : OutputObject {
    while o_r ≠ ⊥ ∧ (size(AL) < k ∨ RSV_r(o_r) ≥ getRSV(AL, k)) do
      /* betrachte das nächste Objekt von der Eingabe (also o_r) */
      SDO := {o_d | ∃rel_d(o_d → o_r)};
        /* alle Objekte, die zu o_r in der gewünschten Beziehung stehen */
      foreach o_d ∈ SDO do
        if ¬contains(AL, o_d) then insert (AL, ⟨o_d; RSV_d(o_d)⟩);
          /* RSV_d muss dazu nach der gewünschten Semantik berechnet werden */
      end /* foreach */;
      o_r := streamGetNext(inputStream);
    end /* while */;
    if o_r = ⊥ ∧ size(AL) < k then
      return ⊥; /* der Eingabestrom ist vollständig abgearbeitet */
    else
      k++;  return getObj(AL, k − 1);
    end /* if */;
  }
}
```

Abbildung 3: Die Klasse *Transferer* in einer programmiersprachlichen Notation

Das Objekt o_r „zu betrachten“, bedeutet konkret die Menge SDO zu bestimmen, die die Objekte vom Typ ot_d enthält, die zu o_r in der gewünschten Beziehung stehen (SDO für „set of desired objects“). In Abbildung 3 wird hierzu die Notation $\exists rel_d(o_d \rightarrow o_r)$ verwendet, die – da wir von o_r ausgehen – deutlich macht, dass die gewünschte Beziehung hier in umgekehrter Richtung zu traversieren ist.

Für jedes Objekt o_d aus SDO, das nicht bereits in AL vorhanden ist, wird der Wert $RSV_d(o_d)$ berechnet und ein Eintrag in AL eingefügt. Die Berechnung von $RSV_d(o_d)$ erfordert dabei mit Ausnahme der Maximumsemantik – die einen Spezi-

alfall darstellt, den wir später genauer betrachten werden –, dass wir wie folgt vorgehen: (1) Die Menge mit allen verbundenen Objekten vom Typ ot_r muss bestimmt werden. (2) Für diese verbundenen Objekte müssen die RSV_r Werte berechnet werden. (3) Nun kann $RSV_d(o_d)$ von diesen RSV_r Werten abgeleitet werden, indem die der gewünschten Semantik entsprechende Funktion $\mathcal{F}$ angewendet wird.

- Falls der Eingabestrom vollständig abgearbeitet ist ($o_r = \perp$) und alle Objekte aus *AL* bereits ausgegeben wurden ($\mathsf{size}(AL) < k$), ist auch der Ausgabestrom des *Transferer*s vollständig erstellt und ein „$\perp$"-Objekt wird zurückgegeben.

- Anderenfalls ist entweder der Eingabestrom vollständig abgearbeitet oder das nächste zu bearbeitende Element aus dem Eingabestrom (o_r) hat einen RSV_r Wert der höchstens so groß ist wie der höchste RSV_d Wert eines noch nicht ausgegebenen Elements in AL (durch die Sortierung in *AL* ist dies Element k in *AL*). In diesem Fall können wir dieses Element von *AL* auf dem Ausgabestrom ausgeben.

  Solange die festgelegte Semantik für RSV_d sicherstellt, dass $RSV_d(do_i) \leq \max\{RSV_r(ro_{i,1}), \ldots, RSV_r(ro_{i,n_i})\}$ gilt, arbeitet das obige Vorgehen korrekt. Diese Bedingung ist offensichtlich für die Maximum-, die Minimum-, die Durchschnitts- und die gewichtete Durchschnittssemantik erfüllt, weil diese Werte den Maximalwert der betrachteten Menge sicher nicht übersteigen können. Auf Basis der obigen Ungleichung können wir, sobald $RSV_r(o_r) < \mathsf{getRSV}(AL, k)$ gilt, sicher sein, dass das k-te Element in AL im weiteren Verlauf des Algorithmus nicht mehr von dieser Position verdrängt wird. Da neue Einträge in *AL* immer hinter bereits vorhandenen Einträgen mit gleichem RSV_d Wert abgelegt werden, gilt dies auch für $RSV_r(o_r) \leq \mathsf{getRSV}(AL, k)$.

  Dies kann wie folgt verdeutlicht werden: Falls o_r nicht das Objekt mit dem höchsten RSV_r Wert ist, das zu einem bestimmten Objekt o_d in Beziehung steht, so wurde o_d bereits betrachtet als das verbundene Objekt mit dem höchsten RSV_r Wert aus dem Eingabestrom entnommen und verarbeitet wurde. Folglich ist, wenn ein Objekt o_d zum ersten Mal betrachtet wird, das verbundene Objekt o_r, das die Betrachtung verursacht, sicher das verbundene Objekt mit dem höchsten RSV_r Wert. Durch die oben angegebene Ungleichung ist dabei sichergestellt, dass der Wert $RSV_d(o_d)$ kleiner oder gleich $RSV_r(o_r)$ ist. Daher können wir das k-te Objekt aus AL auf dem Ausgabestrom ausgeben, sobald $RSV_r(o_r) \leq \mathsf{getRSV}(AL, k)$ gilt, weil in dieser Situation AL sicher die k Objekte mit den höchsten RSV_d Werten enthält.

Wie oben erwähnt, bildet die Maximumsemantik einen Spezialfall, der einige Vereinfachungen erlaubt. Bei dieser Semantik entfällt die Notwendigkeit die RSV_d Werte in der **foreach** Schleife zu berechnen, weil immer dann, wenn noch kein Eintrag für o_d in AL vorhanden ist, o_r sicher das verbundene Objekt mit dem höchsten RSV_r Wert für o_d ist. Folglich gilt $RSV_d(o_d) = RSV_r(o_r)$. Dies hat wiederum zur Folge, dass die Operation $\mathsf{insert}\,(AL, \langle o_d; RSV_d(o_d)\rangle)$ in der getNext-Methode durch die wesentlich effizientere Operation $\mathsf{insert}\,(AL, \langle o_d; RSV_r(o_r)\rangle)$ ersetzt werden kann.

Weitere Vereinfachungsmöglichkeiten ergeben sich, wenn die Beziehung rel_d eine 1:n-Beziehung ist. In diesem Fall ist *SDO* immer eine einelementige Menge.

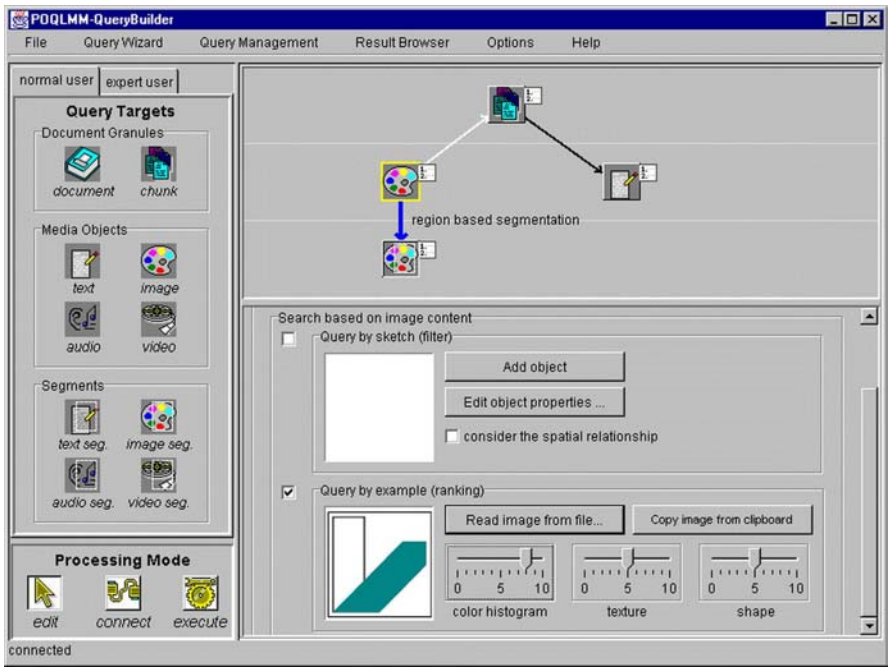

Abbildung 4: Beispiel einer Benutzungsoberfläche, die die Konzepte *Ranker*, *Combiner*, *Transferer* und *Filter* aufgreift

6 Überlegungen zur Benutzungsoberfläche

Aufbauend auf der prototypischen Implementierung unseres Systems (vgl. Abschnitt 7) haben wir eine Benutzungsoberfläche entwickelt, die die *Ranker*, *Combiner*, *Transferer* und *Filter* zur Verfügung stellt und dem Benutzer die Möglichkeit gibt, seine Anfrage durch eine grafische Kombination dieser Komponenten zu definieren. Abbildung 4 zeigt diese Oberfläche, wobei exemplarisch die in Abschnitt 2 beschriebene Anfrage definiert wurde.

Zur Definition einer Anfrage können die Icons aus dem linken Teilfenster auf das Anfragedefinitionsfenster rechts oben gezogen werden. Zu den Icons, die im Wesentlichen die Objekttypen aus unserem Beispielschema repräsentieren, können dann Selektionsbedigungen und Ordnungskriterien definiert werden. In Abbildung 4 ist im unteren rechten Teil exemplarisch das entsprechende Formular für Bildsegmente dargestellt. Bei der Definition der Anfrage können auch die Semantiken für die einzelnen Transferoperationen definiert werden – hierzu existieren entsprechende Popup-Menus.

Bei der Definition zahlreicher Beispielanfragen hat sich die Oberfläche als leistungsfähig und intuitiv erwiesen. Sie macht deutlich, dass die arbeit mit Rankern, Combinern, Transferern, etc. nicht nur auf der Implementierungsebene sondern auch an der Endbenutzerschnittstelle sinnvoll sein kann. Die Details der Oberfläche sind in [HR01a] beschrieben.

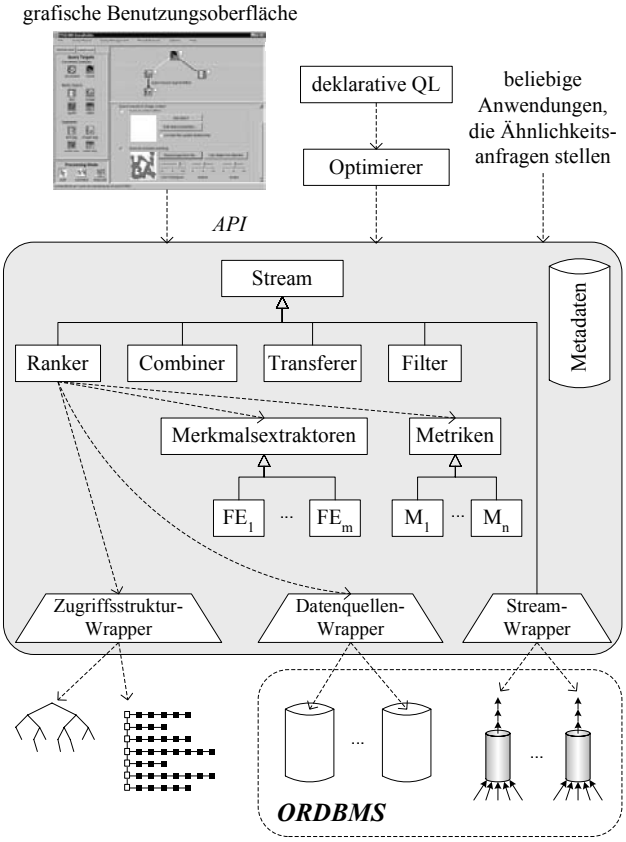

Abbildung 5: Architektur der prototypischen Implementierung

7 Systemarchitektur und experimentelle Ergebnisse

Die Architektur unserer prototypischen Implementierung basiert auf der Idee, die Daten in einem oder mehreren externen Datenspeichern abzulegen. Konkret verwenden wir hierzu eine objektrelationale Datenbank (ORDBMS). Die mit Strömen analog zu Pipes arbeitende *„retrieval engine“* ist in Java oberhalb des Datenspeichers realisiert und stellt eine Programmierschnittstelle (API) zur Verfügung, um die Realisierung ähnlichkeitsbasierter Suchdienste zu ermöglichen. Abbildung 5 verdeutlicht diese Architektur.

Der in Abbildung 5 grau hinterlegte Kern unseres Ansatzes besteht aus vier Hauptteilen: (1) Implementierungen für *Ranker*, *Combiner*, *Transferer* und *Filter*, (2) Implementierungen diverser Methoden für die Extraktion von Merkmalswerten (*feature values*) für verschiedene Objekttypen sowie entsprechender Ähnlichkeitsmaße, (3) einer Komponente zur Verwaltung von Metadaten für das System selbst und die Applikationen, die das System über die API nutzen und (4) Wrappern (oder *Konnektoren*), um externe Datenquellen,

Indexstrukturen und „Stromimplementierungen“ zu integrieren.

Ein **Merkmalsextraktor** erhält ein Objekt eines gegebenen Typs – z.B. ein Bild, ein Textobjekt oder eine Zeitreihe – und extrahiert hieraus einen Merkmalswert für dieses Objekt. Dieser Merkmalswert kann z.B. ein Farbhistogramm, ein Vektor zur Beschreibung der Textureigenschaften oder eine Vektorrepräsentation, die mit Hilfe des Vektorraummodells für einen Text erstellt wurde, sein. Die **Ähnlichkeitsmaße** erhalten zwei Merkmalsrepräsentationen als Eingabe – typischerweise eine, die ein Anfrageobjekt repräsentiert und eine, die ein Objekt aus einer Datenquelle repräsentiert – und bestimmen für dieses Paar einen Retrieval-Status-Wert.

Implementierungen eines *Rankers*, *Combiners*, *Transferers* oder *Filters* werden grundsätzlich von der Klasse **Stream** abgeleitet. Die Schnittstelle dieser Klasse besteht aus einem spezifischen Konstruktor und einer getNext-Methode.

Zum Beispiel erwartet der Konstruktor eines **Rankers** die Spezifikation der Datenquelle – in unserem System typischerweise eine Tabelle des ORDBMS –, einen Merkmalsextraktor, ein Ähnlichkeitsmaß und ein Anfrageobjekt. Der Konstruktor greift dann auf die Metadaten zu und ermittelt, ob für diese Konstellation aus Datenquelle, Merkmalsextraktor und Ähnlichkeitsmaß eine Indexstruktur existiert. In diesem Fall wird die Zugriffstruktur genutzt, um den Ranking-Prozess zu beschleunigen.

Für die Konstruktion eines **Combiners** werden zwei oder mehr Eingabeströme sowie eine entsprechende Gewichtung benötigt. Dabei ist beachtenswert, dass *Combiner* wie Fagin's Algorithmus oder Quick-Combine annehmen, dass auf den Objekten der Eingabeströme ein wahlfreier Zugriff unterstützt wird. Dazu muss der Produzent des Eingabestroms die Möglichkeit bieten, effizient den Retrieval-Status-Wert eines konkreten Objekts zu berechnen, das bisher noch nicht aus dem Eingabestrom gelesen wurde. Der Grund für diese Anforderung ist, das diese Algorithmen immer, wenn sie ein Objekt aus einem ihrer Eingabeströme lesen, versuchen, unmittelbar, den kombinierten Retrieval-Status-Wert für dieses Objekt zu berechnen. Hierzu führen sie einen wahlfreien Zugriff auf den anderen Eingabesrömen aus. Leider bieten einige Implementierungen für die Erzeugung eines Eingabestroms keinen wahlfreien Zugriff. In diesem Fall müssen andere Algorithmen zur Kombination der Eingabeströme verwendet werden – z.B. Nosferatu oder J^*.

Zur Konstruktion eines **Transferers** wird ein Eingabestrom, ein Pfadausdruck zur Definition der gewünschten Beziehung und eine Übertragungssemantik benötigt. Bei unserer Implementierung werden „references“ und „scoped references“, die das unterliegende ORDBMS anbietet, verwendet um die Pfadausdrücke zu definieren. Schließlich sind zur Konstruktion eines **Filters** ein Eingabestrom und eine Filterbedingung erforderlich.

Für jeden erzeugten Strom können nun die Elemente des Ausgabestroms schrittweise über die entsprechende getNext-Methode abgefragt werden.

In der **Metadaten**-Komponente unseres Systems werden Informationen zu den verfügbaren Merkmalsextraktoren, Ähnlichkeitsmaßen, Zugriffsstrukturen, etc. verwaltet. Diese Metadaten, die im System zur Anfrageoptimierung verwendet werden, können auch über die API zugegriffen werden. Hier können die Metadaten z.B. genutzt werden, um die Anfragekonstruktion in einer graphischen Benutzungsoberfläche zu steuern.

Wrapper für Datenspeicher werden benötigt, um Systeme, die die Objekte selbst verwalten, an unser Retrieval-System zu koppeln. Gegenwärtig existiert ein Wrapper zur Anbindung objektrelationaler Datenbanken über JDBC. Wrapper für Zugriffsstrukturen können genutzt werden, um Strukturen, die ursprünglich nicht für das System entwickelt wurden, zu integrieren. Zum Beispiel haben wir eine in C++ geschriebene Implementierung des LSD^h-Baumes über einen entsprechenden Wrapper angebunden. Schließlich können Wrapper für externe Stream-Implementierungen genutzt werden, um Komponenten zu integrieren, die Rangordnungen über Objekten erzeugen. Gegenwärtig wird das Textmodul des unterliegenden ORDBMS über einen solchen Wrapper eingebunden. Dabei stellt eine externe Stream-Implementierung nicht nur ein Ranking, sondern auch den Zugriff auf die verwalteten Objekte selbst zur Verfügung, während eine externe Zugriffsstruktur nur die Merkmalswerte und zugeordnete Objektreferenzen kennt.

Oberhalb der vom System zur Verfügung gestellten API können vielfältige Anwendungen realisiert werden. Ein Beispiel ist die graphische Benutzungsoberfläche aus Abschnitt 6. Ein anderes Beispiel ist die Implementierung einer deklarativen Anfragesprache auf Basis der API. Gegenwärtig arbeiten wir an einer entsprechenden Anpassung unserer Anfragesprache POQL$^{\mathrm{MM}}$ [Hen96, HR01b].

Um die Performance des vorgestellten Ansatzes zu überprüfen, haben wir die Bausteine, wie *Ranker*, *Combiner* und *Transferer*, in Java implementiert. Zur Verwaltung der Metadaten wird ein ORDBMS eingesetzt. Die hier vorgestellten Testergebnisse wurden mit Hilfe einer Dokumentenkollektion ermittelt, die Artikel einer Computerzeitschrift als strukturierte Dokumente enthält. Die Kollektion umfasst 2213 Artikel, 29414 Textblöcke, 4999 Bilder und 19980 Bildsegmente. Diese Dokumente wurden in das unserem System zugrunde liegende ORDBMS eingefügt. Ferner wurden zwei LSD^h-Bäume für Feature-Vektoren erzeugt, die die Farbcharakteristika und die Texturcharakteristika der Bildsegmente verwalten. Für die Farbähnlichkeit wurden zehndimensionale Farbhistogramme nach dem Munsell-Modell verwendet[SW95]. Für die Texturähnlichkeit wurden vierdimensionale Eigenschaftsvektoren genutzt, die die Homogenität, die Energie, den Kontrast und die Entropie der Bildsegmente beschreiben.

Wir haben drei unterschiedliche Arten von Anfragen auf diesen Testdaten ausgeführt: Anfragen, die nur einen *Ranker* nutzen, Anfragen, die zwei *Ranker* und einen darauf aufbauenden *Combiner* nutzen, und Anfragen, die zwei *Ranker*, einen *Combiner* und einen *Transferer* verwenden. Im letzen Fall wurde mit der Durchschnitts- und der Maximumsematik gearbeitet.

Mit der ersten Anfrage wurde nach den k im Hinblick auf die Farbähnlichkeit ähnlichsten Bildsegmenten, verglichen mit einem gegebenen Anfragebild, gesucht. Folglich wurde zur Bearbeitung dieser Anfrage genau ein *Ranker* benötigt. Abbildung 6a zeigt, dass die Performance unseres Systems in diesem Fall weit besser ist als wenn die Anfrage direkt mit den Möglichkeiten des ORDBMS ausgeführt wird. Dabei ist anzumerken, dass die für das ORDBMS angegebenen Werte die besten Werte sind, die sich nach dem Austesten verschiedenster SQL-Anfragen und Indexstrukturen auf der Datenbank ergaben.

In der zweiten Anfrage haben wir nach den k ähnlichsten Bildsegmenten verglichen mit einem Anfragebild gesucht, wobei nun sowohl die Farb- als auch die Texturähnlichkeit

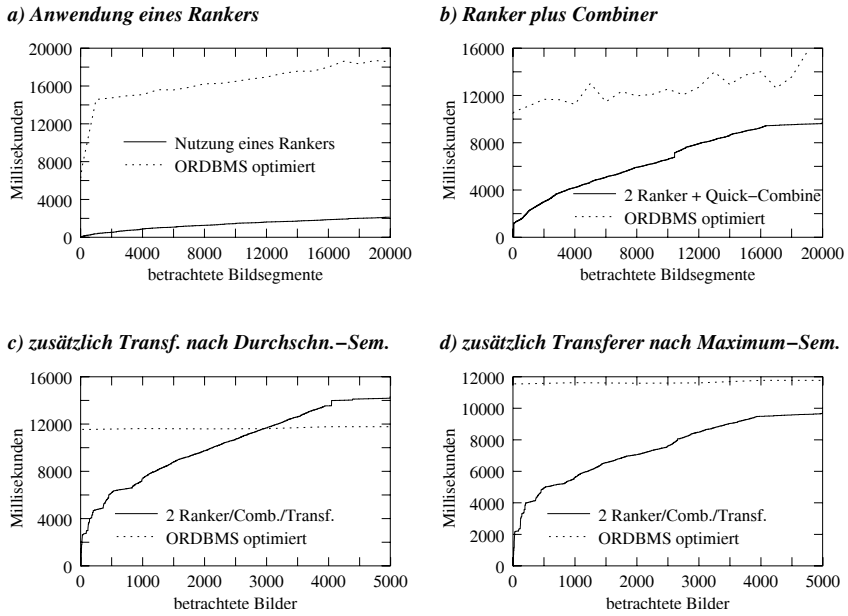

Abbildung 6: Messergebnisse für verschiedene Anfragen

betrachtet wurde. An der Ausführung dieser Anfrage in unserem Ansatz waren folglich zwei *Ranker* und ein *Combiner* (Quick-Combine) beteiligt. Abbildung 6b zeigt, dass auch bei dieser Anfrage unser Ansatz deutlich bessere Ergebnisse liefert als eine optimierte SQL-Anfrage auf dem Datenbanksystem. Auffällig sind die schwankenden Werte bei den Zeitmessungen für die Laufzeiten bzgl. des ORDBMS. Diese Schwankungen liegen darin begründet, dass sich die Messwerte für das ORDBMS durch die Mittelung der Laufzeitergebnisse von jeweils fünf Durchläufen an ausgewählten Messpunkten ergaben.

Schließlich wurde in der dritten Anfrage zusätzlich eine Übertragung des Rankings für die Bildsegmente auf die zugehörigen Bilder durchgeführt. Dazu wurde der in dieser Arbeit beschriebene *RSV-Transfer*-Algorithmus einmal mit der Durchschnittssemantik und einmal mit der Maximumsemantik angewendet. Die Abbildungen 6c und 6d zeigen die entsprechenden Ergebnisse. Es wird deutlich, dass selbst bei der für die Performance eher ungünstigen Durchschnittssemantik für realistische Werte von k eine deutliche Steigerung der Performance gegenüber dem ORDBMS erzielt werden konnte.

Bei der Interpretation der dargestellten Messergebnisse ist zu berücksichtigen, dass unsere Prototyp-Implementierung in Java sicher noch weiteres Optimierungspotential lässt. Trotzdem deuten schon diese Ergebnisse an, dass die Nutzung unseres Ansatzes mit expliziten Komponenten für die Übertragung eines Rankings eine konkurrenzfähige Performance erzielen kann.

8 Zusammenfassung und Ausblick

In der vorliegenden Arbeit haben wir einen expliziten Ansatz zur Übertragung einer Rangordnung, die für Objekte eines bestimmten Typs definiert ist, auf verbundene Objekte vorgestellt. Die explizite Betrachtung dieses Transfer-Schrittes im Rahmen einer komplexen Ähnlichkeitsanfrage hat den Vorteil, dass die Semantik der Übertragung und der entsprechende Algorithmus allgemeingültig betrachtet werden können. Eine auf diesem Ansatz basierende Benutzungsoberfläche macht deutlich, dass er auch zur graphischen Formulierung von Anfragen geeignet ist und eine prototypische Implementierung zeigt, dass die erreichbare Performance konkurrenzfähig ist.

Der in dieser Arbeit vorgeschlagene Ansatz optimiert dabei die Anfragebearbeitung bewusst im Hinblick auf Ranking-Bedingungen und die Nutzung entsprechender Zugriffsstrukturen und Algorithmen. Im Zusammenhang einer objektrelationalen Datenbank kann er deshalb auch im Sinne eines weiteren möglichen Ausführungsplans in den Optimierungsprozess mit einbezogen werden. Die Überlegungen hierzu sind Gegenstand unserer aktuellen Forschung. Dies gilt auch für Betrachtungen zur Ergebnisqualität, in deren Rahmen auf Basis von Benutzerstudien ermittelt werden soll, inwieweit strukturierte Ähnlichkeitsanfragen – wie die im Papier betrachtete Beispielanfrage – eine bessere Erfüllung der Informationswünsche der Benutzer erlauben als Ähnlichkeitsanfragen, die sich nur auf die Objekte selbst beziehen.

Literaturverzeichnis

[AC97] Anastasia Analyti and Stavros Christodoulakis. Content-Based Querying. In P.M.G. Apers, H.M. Blanken, and M.A.W. Houtsma, editors, *Multimedia Databases in Perspektive*, chapter 8, pages 145–180. Springer, Berlin, 1997.

[BCG02] Nicolas Bruno, Surajit Chaudhuri, and Luis Garvano. Top-*k* selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transactions on Database Systems*, 27(2):153–187, 2002.

[BKK96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree : An Index Structure for High-Dimensional Data. In *VLDB'96, Proc. 22th Intl. Conf. on Very Large Data Bases*, pages 28–39, Mumbai (Bombay), India, September 1996.

[BL85] Chris Buckley and A.F. Lewit. Optimization of inverted vector searches. In *Proc. of the 8th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 97–105, New York, USA, 1985.

[CK97] Michael J. Carey and Donald Kossmann. On Saying "Enough Already!" in SQL. In *Proc. of the 1997 ACM SIGMOD Intl. Conf. on Management of Data*, pages 219–230, Tucson, Arizona, 13–15 June 1997.

[CK98] Michael J. Carey and Donald Kossmann. Reducing the Braking Distance of an SQL Query Engine. In *VLDB'98, Proc. 24th Intl. Conf. on Very Large Data Bases*, pages 158–169, New York, USA, 1998.

[Fag98] Ronald Fagin. Fuzzy Queries in Multimedia Database Systems. In *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–10, Seattle, Washington, 1998. ACM Press.

[FLN01] Ronald Fagin, Ammon Lotem, and Moni Naor. Optimal Aggregation Algorithms for Middleware. In *Proc. of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Santa Barbara, California, USA, May 2001.

[GBK00] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Optimizing Multi-Feature Queries for Image Databases. In *VLDB 2000, Proc. 26th Intl. Conf. on Very Large Data Bases*, pages 419–428, Cairo, Egypt, September 2000.

[GBK01] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Towards Efficient Multi-Feature Queries in Heterogeneous Environments. In *Intl. Conf. on Information Technology: Coding and Computing (ITCC 2001)*, pages 622–628, Las Vegas, USA, 2001.

[Hen96] Andreas Henrich. Document Retrieval Facilities for Repository-Based System Development Environments. In *Proc. of the 19th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 101–109, Zürich, 1996.

[Hen98] Andreas Henrich. The LSD$^{\mathrm{h}}$-Tree: An Access Structure for Feature Vectors. In *Proc. of the 14th Intl. Conf. on Data Engineering, February, 1998, Orlando, Florida*, pages 362–369. IEEE Computer Society, 1998.

[HR01a] Andreas Henrich and Günter Robbert. An End User Retrieval Interface for Structured Multimedia Documents. In *Proc. 7th Workshop on Multimedia Information Systems, MIS'01*, pages 71–80, Capri, Italy, 2001.

[HR01b] Andreas Henrich and Günter Robbert. POQL$^{\mathrm{MM}}$: A Query Language for Structured Multimedia Documents. In *Proc. 1st Intl. Workshop on Multimedia Data and Document Engineering, MDDE'01*, pages 17–26, Lyon, France, 2001.

[IAE02] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Joining Ranked Inputs in Practice. In *VLDB'02, Proc. of 28th Intl. Conf. on Very Large Data Bases*, Hong Kong, China, 2002.

[NCS+01] Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting Incremental Join Queries on Ranked Inputs. In *VLDB 2001, Proc. of 27th Intl. Conf. on Very Large Data Bases*, pages 281–290, Roma, Italy, 9 2001.

[NR99] Surya Nepal and M. V. Ramakrishna. Query Processing Issues in Image (Multimedia) Databases. In *Proc. of the 15th Intl. Conf. on Data Engineering*, pages 22–29, Sydney, Austrialia, 1999. IEEE Computer Society.

[PP97] Ulrich Pfeifer and Stefan Pennekamp. Incremental Processing of Vague Queries in Interactive Retrieval Systems. In *HIM '97, Proc. Hypertext - Information Retrieval - Multimedia '97*, pages 223–235, Dortmund, 1997. Universitätsverlag Konstanz.

[Sal89] G. Salton. *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, Reading, Mass., 1989.

[SMMR99] D. M. Squire, W. Müller, H. Müller, and J. Raki. Content-based query of image databases, inspirations from text retrieval: inverted files, frequency-based weights and relevance feedback. In *The 11th Scandinavian Conf. on Image Analysis (SCIA 99)*, pages 143–149, Kangerlussuaq, Greenland, 1999.

[SW95] J. Sturges and T.W.A. Whitfield. Locating basic colours in the munsell space. *Color Research and Application*, 20:364–376, 1995.

[VGJL94] E. M. Voorhees, N. K. Gupta, and B. Johnson-Laird. The Collection Fusion Problem. In *Proc. of the 3rd Text Retrieval Conf. (TREC-3)*, pages 95–104, Gaithersburg, Maryland, 11 1994. NIST Special Publication 500 - 225.

[WSB98] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB'98, Proc. of 24rd Intl. Conf. on Very Large Data Bases*, pages 194–205, New York, 1998.

[ZSAR98] Pavel Zezula, Pasquale Savino, Giuseppe Amato, and Fausto Rabitti. Approximate Similarity Retrieval with M-Trees. *VLDB Journal*, 7(4):275–293, 1998.